RANSAC Runtime Analysis

Main code creates it’s core threads in

```
for i := 0; i < iterations; i++ {
        c5[i] = SupportingPointFinder(points, eps, c4)
    }
```

This determines its runtime.
—---

Changing c5 value from:

```
c5 := make([]<-chan Plane3DwSupport, iterations)
```

To:

```
c5 := make(<-chan chan Plane3DwSupport, iterations)
```

Creates additional runtime, averaging to 400ms +, rather than its usual 80ms - 90ms trend from running an array of channels to open.

The amount of goroutines created depends on the iteration amount, as that is the only amount necessary to be created, thus the iteration values are left unchanged. Decreasing iteration values creates a faster runtime, but does not display the correct output result required. Thus there is direct correlation, but incorrect input.

—--
-

FanIn function only loops until each function is taken in. As the previous function stops directly after creating a singular channel, no delay in input is created, causing it to be time-efficient.

—--
-

Back to the initial SupportingFinder, if it is changed to now include a an additional integer value, that would help in storing more channels inside its function, we start seeing consistent results as the iterations are decreased.

This function,

```go
func SupportingPointFinder(pc []Point3D, eps float64, in <-chan Plane3D)
<-chan Plane3DwSupport {
    out := make(chan Plane3DwSupport, 1)
    go func() {
        defer close(out)
        out <- GetSupport(<-in, pc, eps)
    }()
    return out
}
```

Becomes:

```go
func SupportingPointFinder(x int, pc []Point3D, eps float64, in <-chan
Plane3D) <-chan Plane3DwSupport {
    out := make(chan Plane3DwSupport, x)
    go func() {
        defer close(out)
        for i := 0; i < x; i++ {
            out <- GetSupport(<-in, pc, eps)
        }

    }()
    return out
}
```

And we change the main function to become:

```go
x := 5

    c5 := make([]<-chan Plane3DwSupport, iterations/x)

    for i := 0; i < iterations/x; i++ {
        c5[i] = SupportingPointFinder(x, points, eps, c4)
    }
```
Here are the runtime averages, in comparison to x:

x:	Runtime average:
1	88
2	87
3	86
4	86
5	84
...	...
10	90

The lowest results achieved are by balancing x at the value 5 **at a runtime observed of 82**, creating about iteration/5 channels, each with x number of inputs inside the channel to later fan in. This creates the most balanced output comparatively.

FanIn function is not affected as it still loops the same amount of inputs (iteration/5 * 5)